Quantum Predicative Programming

Anya Tafliovich and E.C.R. Hehner

University of Toronto

Abstract. The subject of this work is quantum predicative programming — the study of developing of programs intended for execution on a quantum computer. We look at programming in the context of formal methods of program development, or programming methodology. Our work is based on probabilistic predicative programming, a recent generalisation of the well-established predicative programming. It supports the style of program development in which each programming step is proven correct as it is made. We inherit the advantages of the theory, such as its generality, simple treatment of recursive programs, time and space complexity, and communication. Our theory of quantum programming provides tools to write both classical and quantum specifications, develop quantum programs that implement these specifications, and reason about their comparative time and space complexity all in the same framework.

1 Introduction

Modern physics is dominated by concepts of quantum mechanics. Today, over seventy years after its recognition by the scientific community, quantum mechanics provides the most accurate known description of nature's behaviour. Surprisingly, the idea of using the quantum mechanical nature of the world to perform computational tasks is very new, less than thirty years old. Quantum computation and quantum information is the study of information processing and communication accomplished with quantum mechanical systems. In recent years the field has grown immensely. Scientists from various fields of computer science have discovered that thinking physically about computation yields new and exciting results in computation and communication. There has been extensive research in the areas of quantum algorithms, quantum communication and information, quantum cryptography, quantum error-correction, adiabatic computation, measurement-based quantum computation, theoretical quantum optics, and the very new quantum game theory. Experimental quantum information and communication has also been a fruitful field. Experimental quantum optics, ion traps, solid state implementations and nuclear magnetic resonance all add to the experimental successes of quantum computation.

The subject of this work is quantum programming — the study of developing programs intended for execution on a quantum computer. We assume a model of a quantum computer proposed by Knill [24]: a classical computer with access to a quantum device that is capable of storing quantum bits, performing certain operations and measurements on these bits, and reporting the results of the measurements.

We look at programming in the context of formal methods of program development, or programming methodology. This is the field of computer science concerned with applications of mathematics and logic to software engineering tasks. In particular, the formal methods provide tools to formally express software specifications, prove correctness of implementations, and reason about various properties of specifications (e.g. implementability) and implementations (e.g. time and space complexity). Today formal methods are successfully employed in all stages of software development, such as requirements elicitation and analysis, software design, and software implementation.

In this work the theory of quantum programming is based on probabilistic predicative programming, a recent generalisation of the well-established predicative programming [19,20], which we deem to be the simplest and the most elegant programming theory known today. It supports the style of program development in which each programming step is proven correct as it is made. We inherit the advantages of the theory, such as its generality, simple treatment of recursive programs, and time and space complexity. Our theory of quantum programming provides tools to write both classical and quantum specifications, develop quantum programs that implement these specifications, and reason about their comparative time and space complexity all in the same framework.

The rest of this work is organised as follows. Section 2.1 is the introduction to quantum computation. It assumes that the reader has some basic knowledge of linear algebra and no knowledge of quantum computing. Section 2.2 contains the introduction to probabilistic predicative programming. The reader is assumed to have some background in logic, but no background in programming theory is necessary. The contribution of this work is section 3 which defines the quantum system, introduces programming with the quantum system, and several well-known problems, their classical and quantum solutions, and their formal comparative time complexity analyses. Section 4 states conclusions and outlines directions for future research.

1.1 Related work

Traditionally, quantum computation is presented in terms of quantum circuits. Recently, there has been an attempt to depart from this convention for the same reason that classical computation is generally not presented in terms of classical circuits. As we develop more complex quantum algorithms, we will need ways to express higher-level concepts with control structures in a readable fashion.

In 2000 Ömer [28] introduced the first quantum programming language QCL. Following his work, Bettelli *et. al.* developed a quantum programming language with syntax based on C++. These two works did not involve any verification techniques.

Sanders and Zuliani in [29] introduced a quantum language qGCL, which is an extension of pGCL [26], which in turn generalises Dijkstra's guarded-command language to include probabilism. Zuliani later extends this attempt at formal program development and verification in [36], which discusses treatment of non-determinism in quantum programs, and in [38], where the attempt is made to build on Aharonov's work to reason about mixed states computations. Zuliani also provides tools to approach the task of compiling quantum programs in [37].

A large amount of work in the area was performed in the past two years. In [4], [25], and [22] process algebraic approaches were explored. Tools developed in the field of category theory were successfully employed by [1], [2], [3], [11], [30], and others to reason about quantum computation. In [7] and [8] a functional language with semantics in a form of a term rewrite system is introduced and a notion of linearity and how it pertains to quantum systems are examined. A functional language QML with design guided by its categorical semantics is defined in [5]. Following on this work, [6] provides a sound and complete equational theory for QML. Weakest preconditions appropriate for quantum computation are introduced in [15]. This work is interesting, in part, because it diverts from the standard approach of reducing quantum computation to probabilistic one. It also provides semantics for the language of [30]. Other interesting work by the same authors include reasoning about knowledge in quantum systems ([16]) and developing a formal model for distributed measurement-based quantum computation ([12]). A similar work is introduced in [17], where a language CQP for modelling communication in quantum systems is defined. The latter approaches have an advantage over process algebraic approaches mentioned earlier in that they explicitly allow a quantum state to be transmitted between processes. Building of the work of [31], [33] defines a higher order quantum programming language based on a linear typed lambda calculus, which is similar to the work of [34].

1.2 Our contribution

Our approach to quantum programming amenable to formal analysis is very different from almost all of those described above. Work of [29], [36], [38] is the only one which is similar to our work. The contribution of this paper is twofold. Firstly, by building our theory on that in [20], we inherit the advantages it offers. The definitions of specification and program are simpler: a specification is a boolean (or probabilistic) expression and a program is a specification. The treatment of recursion is simple: there is no need for additional semantics of loops. The treatment of termination simply follows from the introduction of a time variable; if the final value of the time variable is ∞, then the program is a non-terminating one. Correctness and time and space complexity are proved in the same fashion; moreover, after proving them separately, we naturally obtain the conjunction. Secondly, the way Probabilistic Predicative Programming is extended to Quantum Predicative Programming is simple and intuitive. The use of Dirac-like notation makes it easy to write down specifications and develop algorithms. The treatment of computation with mixed states does not require any additional mechanisms. Quantum Predicative Programming fully preserves Predicative Programming's treatment of parallel programs and communication, which provides for a natural extension to reason about quantum communication protocols, such as BB84 ([9]), distributed quantum algorithms, such as distributed Shor's algorithm ([35]), as well as their time, space, and entanglement complexity.

2 Preliminaries

2.1 Quantum Computation

In this section we introduce the basic concepts of quantum mechanics, as they pertain to the quantum systems that we will consider for quantum computation. The discussion of the underlying physical processes, spin-$\frac{1}{2}$-particles, etc. is not of our interest. We are concerned with the model for quantum computation only. A reader not familiar with quantum computing can consult [27] for a comprehensive introduction to the field.

The *Dirac notation*, invented by Paul Dirac, is often used in quantum mechanics. In this notation a vector v (a column vector by convention) is written inside a *ket*: $|v\rangle$. The dual vector of $|v\rangle$ is $\langle v|$, written inside a *bra*. The inner products are *bra-kets* $\langle v|w\rangle$. For n-dimensional vectors $|u\rangle$ and $|v\rangle$ and m-dimensional vector $|w\rangle$, the value of the inner product $\langle u|v\rangle$ is a scalar and the outer product operator $|v\rangle\langle w|$ corresponds to an m by n matrix. The Dirac notation clearly distinguishes vectors from operators and scalars, and makes it possible to write operators directly as combinations of bras and kets.

In quantum mechanics, the vector spaces of interest are the Hilbert spaces of dimension 2^n for some $n \in \mathbb{N}$. A convenient orthonormal basis is what is called a *computational basis*, in which we label 2^n basis vectors using binary strings of length n as follows: if s is an n-bit string which corresponds to the number x_s, then $|s\rangle$ is a 2^n-bit (column) vector with 1 in position x_s and 0 everywhere else. The tensor product $|i\rangle \otimes |j\rangle$ can be written simply as $|ij\rangle$. An arbitrary vector in a Hilbert space can be written as a weighted sum of the computational basis vectors.

Postulate 1 (state space) Associated to any isolated physical system is a Hilbert space, known as the *state space* of the system. The system is completely described by its *state vector*, which is a unit vector in the system's state space.

Postulate 2 (evolution) The evolution of a closed quantum system is described by a *unitary transformation.*

Postulate 3 (measurement) Quantum measurements are described by a collection $\{M_m\}$ of *measurement operators*, which act on the state space of the system being measured. The index m refers to the possible measurement outcomes. If the state of the system immediately prior to the measurement is described by a vector $|\psi\rangle$, then the probability of obtaining result m is $\langle\psi|M_m^\dagger M_m|\psi\rangle$, in which case the state of the system immediately after the measurement is described by the vector $\frac{M_m|\psi\rangle}{\sqrt{\langle\psi|M_m^\dagger M_m|\psi\rangle}}$. The measurement operators satisfy the *completeness equation* $\sum m \cdot M_m^\dagger M_m = I$.

An important special class of measurements is *projective measurements*, which are equivalent to general measurements provided that we also have the ability to perform unitary transformations.

A projective measurement is described by an *observable* M, which is a Hermitian operator on the state space of the system being measured. This observable has a spectral decomposition $M = \sum m \cdot \lambda_m \times P_m$, where P_m is the projector onto the eigenspace of M with eigenvalue λ_m, which corresponds to the outcome of the measurement. The probability of measuring m is $\langle\psi|P_m|\psi\rangle$, in which case immediately after the measurement the system is found in the state $\frac{P_m|\psi\rangle}{\sqrt{\langle\psi|P_m|\psi\rangle}}$.

Given an orthonormal basis $|v_m\rangle$, $0 \leq m < 2^n$, measurement with respect to this basis is the corresponding projective measurement given by the observable $M = \sum m \cdot \lambda_m \times P_m$, where the projectors are $P_m = |v_m\rangle\langle v_m|$.

Measurement with respect to the computational basis is the simplest and the most commonly used class of measurements. In terms of the basis $|m\rangle$, $0 \leq m < 2^n$, the projectors are $P_m = |m\rangle\langle m|$ and $\langle\psi|P_m|\psi\rangle = |\psi_m|^2$. The state of the system immediately after measuring m is $|m\rangle$.

In the case of a single qubit, for example, measurement of the state $\alpha \times |0\rangle + \beta \times |1\rangle$ results in the outcome 0 with probability $|\alpha|^2$ and outcome 1 with probability $|\beta|^2$. The state of the system immediately after the measurement is $|0\rangle$ or $|1\rangle$, respectively.

Suppose the result of the measurement is ignored and we continue the computation. In this case the system is said to be in a *mixed state*. A mixed state is not the actual physical state of the system. Rather it describes our knowledge of the state the system is in. In the above example, the mixed state is expressed by the equation $|\psi\rangle = |\alpha|^2 \times \{|0\rangle\} + |\beta|^2 \times \{|1\rangle\}$. The equation is meant to say that $|\psi\rangle$ is $|0\rangle$ with probability $|\alpha|^2$ and it is $|1\rangle$ with probability $|\beta|^2$. An application of operation U to the mixed state results in another mixed state, $U(|\alpha|^2 \times \{|0\rangle\} + |\beta|^2 \times \{|1\rangle\}) = |\alpha|^2 \times \{U|0\rangle\} + |\beta|^2 \times \{U|1\rangle\}$.

Postulate 4 (composite systems) The state space of a composite physical system is the tensor product of the state spaces of the component systems. If we have systems numbered 0 up to and excluding n, and each system i, $0 \leq i < n$, is prepared in the state $|\psi_i\rangle$, then the joint state of the composite system is $|\psi_0\rangle \otimes |\psi_1\rangle \otimes \ldots \otimes |\psi_{n-1}\rangle$.

While we can always describe a composite system given descriptions of the component systems, the reverse is not true. Indeed, given a state vector that describes a composite system, it may not be possible to factor it to obtain the state vectors of the component systems. A well-known example is the state $|\psi\rangle = |00\rangle/\sqrt{2} + |11\rangle/\sqrt{2}$. Such state is called an *entangled* state.

2.2 Probabilistic Predicative Programming

This section introduces the programming theory of our choice, on which our work on quantum programming is based — probabilistic predicative programming. We briefly introduce parts of the theory necessary for understanding section 3 of this work. For a course in predicative programming the reader is referred to [19]. Introduction to probabilistic predicative programming can be found in [20].

Predicative programming In predicative programing a specification is a boolean expression. The variables in a specification represent the quantities of interest, such as prestate (inputs), poststate (outputs), and computation time and space. We use primed variables to describe outputs and unprimed variables to describe inputs. For example, specification $x' = x + 1$ in one integer variable x states that the final value of x is its initial value plus 1. A computation *satisfies* a specification if, given a prestate, it produces a poststate, such that the pair makes the specification true. A specification is *implementable* if for each input state there is at least one output state that satisfies the specification.

We use standard logical notation for writing specifications: $\wedge$ (conjunction), $\vee$ (disjunction), $\Rightarrow$ (logical implication), $=$ (equality, boolean equivalence), $\neq$ (non-equality, non-equivalence), and **if then else**. $==$ and $\Longrightarrow$ are the same as $=$ and $\Rightarrow$, but with lower precedence. We use standard mathematical notation, such as $+\ -\ *\ /\ mod$. We use lowercase letters for variables of interest and uppercase letters for specifications.

In addition to the above, we use the following notations: σ (prestate), σ' (poststate), ok ($\sigma' = \sigma$), and $x := e$ ($x' = e \wedge y' = y \wedge \ldots$). ok specifies that the values of all variables are unchanged. In the assignment $x := e$, x is a state variable (unprimed) and e is an expression (in unprimed variables) in the domain of x.

If R and S are specifications in variables $x, y, \ldots$, R'' is obtained from R by substituting all occurrences of primed variables $x', y', \ldots$ with double-primed variables $x'', y'', \ldots$, and S'' is obtained from S by substituting all occurrences of unprimed variables $x, y, \ldots$ with double-primed variables $x'', y'', \ldots$, then the *sequential composition* of R and S is defined by

$$R; S == \exists x'', y'', \ldots \cdot R'' \wedge S''$$

.

Various laws can be proven about sequential composition. One of the most important ones is the substitution law, which states that for any expression e of the prestate, state variable x, and specification P,

$$x := e; P == \text{(for } x \text{ substitute } e \text{ in } P\text{)}$$

Specification S *is refined by* specification P if and only if S is satisfied whenever P is satisfied:

$$\forall \sigma, \sigma' \cdot S \Leftarrow P$$

Specifications S and P are equal if and only if they are satisfied simultaneously:

$$\forall \sigma, \sigma' \cdot S = P$$

Given a specification, we are allowed to implement an equivalent specification or a stronger one.

Informally, a *bunch* is a collection of objects. It is different from a set, which is a collection of objects in a package. Bunches are simpler than sets; they don't have a nesting structure. See [20] for an introduction to bunch theory. A bunch of one element is the element itself. We use upper-case to denote arbitrary bunches and lower-case to denote elements (an element is the same as a bunch of one element). A, B denotes the union of bunches A and B. $A : B$ denotes bunch inclusion — bunch A is included in bunch B. We use notation $x, ..y$ to mean from (including) x to (excluding) y.

If x is a fresh (previously unused) name, D is a bunch, and b is an arbitrary expression, then $\lambda x : D \cdot b$ is a *function* of a variable (parameter) x with domain D and body b. If f is a function, then Δf denotes the domain of f. If $x : \Delta f$, then fx (f applied to x) is the corresponding element in the range. A function of n variables is a function of 1 variable, whose body is a function of $n-1$ variables, for $n > 0$. A predicate is function whose body is a boolean expression. A relation is a function whose body is a predicate. A higher-order function is a function whose parameter is a function.

A *quantifier* is a unary prefix operator that applies to functions. If p is a predicate, then $\forall p$ is the boolean result, obtained by first applying p to all the elements in its domain and then taking the conjunction of those results. Taking the disjunction of the results produces $\exists p$. Similarly, if f is a numeric function, then $\sum f$ is the numeric result, obtained by first applying f to all the elements in its domain and then taking the sum of those results.

For example, applying the quantifier $\sum$ to the function $\lambda i : 0, ..2^n \cdot |\psi i|^2$, for some function ψ, yields: $\sum \lambda i : 0, ..2^n \cdot |\psi i|^2$, which for the sake of simplicity we abbreviate to $\sum i : 0, ..2^n \cdot |\psi i|^2$. In addition, we allow a few other simplifications. For example, we can omit the domain of a variable if it is clear from the context. We can also group variables from several quantifications. For example, $\sum i : 0, ..2^n \cdot \sum j : 0, ..2^n \cdot 2^{-m-n}$ can be abbreviated to $\sum i, j : 0, ..2^n \cdot 2^{-m-n}$.

A *program* is an implemented specification. For simplicity we only take the following to be implemented: *ok*, assignment, **if then else**, sequential composition, booleans, numbers, bunches, and functions.

Given a specification S, we proceed as follows. If S is a program, there is no work to be done. If it is not, we build a program P, such that P refines S, i.e. $S \Leftarrow P$. The refinement can proceed in steps: $S \Leftarrow \ldots \Leftarrow R \Leftarrow Q \Leftarrow P$.

One of the best features of Hehner's theory, is its simple treatment of recursion. In $S \Leftarrow P$ it is possible for S to appear in P. No additional rules are required to prove the refinement. For example, it is trivial to prove that

$$x \geq 0 \Rightarrow x' = 0 \Longleftarrow \textbf{ if } x = 0 \textbf{ then } ok \textbf{ else } (x := x - 1; x \geq 0 \Rightarrow x' = 0)$$

The specification says that if the initial value of x is non-negative, its final value must be 0. The solution is: if the value of x is zero, do nothing, otherwise decrement x and repeat.

How long does the computation take? To account for time we add a time variable t. We use t to denote the time, at which the computation starts, and t' to denote the time, at which the computation ends. In case of non-termination, $t' = \infty$. This is the only characteristic by which we distinguish terminating programs from non-terminating ones. See [21] for a discussion on treatment of termination. We choose to use a *recursive time* measure, in which we charge 1 time unit for each time P is called. We replace each call to P to include the time increment as follows:

$$P \Longleftarrow \textbf{ if } x = 0 \textbf{ then } ok \textbf{ else } (x := x - 1; t := t + 1; P)$$

It is easy to see that t is incremented the same number of times that x is decremented, i.e. $t' = t + x$, if $x \geq 0$, and $t' = \infty$, otherwise. Just as above, we can prove:

$$\begin{aligned} &x \geq 0 \wedge t' = t + x \vee x < 0 \wedge t' = \infty \\ \Longleftarrow\ &\textbf{if } x = 0 \textbf{ then } ok \\ &\quad \textbf{else } (x := x - 1;\ t := t + 1;\ x \geq 0 \wedge t' = t + x \vee x < 0 \wedge t' = \infty) \end{aligned}$$

Probabilistic predicative programming Probabilistic predicative programming was introduced in [19] and was further developed in [20]. It is a generalisation of predicative programming that allows reasoning about probability distributions of values of variables of interest. Although in this work we apply this reasoning to boolean and integer variables only, the theory does not change if we want to work with real numbers: we replace summations with integrals.

A *probability* is a real number between 0 and 1, inclusive. A *distribution* is an expression whose value is a probability and whose sum over all values of variables is 1. For example, if n is a positive natural variable, then 2^{-n} is a distribution, since for any n, 2^{-n} is a probability, and $\sum n \cdot 2^{-n} = 1$. In two positive natural variables m and n, 2^{-n-m} is also a distribution. If a distribution of several variables can be written as a product of distributions of the individual variables, then the variables are *independent*. For example, m and n in the previous example are independent. Given a distribution of several variables, we can sum out some of the variables to obtain a distribution of the rest of the variables. In our example, $\sum n \cdot 2^{-n-m} = 2^{-m}$, which is a distribution of m.

To generalise boolean specifications to probabilistic specifications, we use 1 and 0 for boolean *true* and *false*, respectively.[1] If S is an implementable deterministic specification and p is a distribution of the initial state $x, y, ...$, then the distribution of the final state is

$$\sum x, y, ... \cdot S \times p$$

[1] Readers familiar with $\top$ and $\bot$ notation can notice that we take the liberty to equate $\top = 1$ and $\bot = 0$.

If R and S are specifications in variables $x, y, \ldots$, R'' is obtained from R by substituting all occurrences of primed variables $x', y', \ldots$ with double-primed variables $x'', y'', \ldots$, and S'' is obtained from S by substituting all occurrences of unprimed variables $x, y, \ldots$ with double-primed variables $x'', y'', \ldots$, then the *sequential composition* of R and S is defined by

$$R;S = \sum x'', y'', \ldots \cdot R'' \times S''$$

If p is a probability and R and S are distributions, then

$$\textbf{if } p \textbf{ then } R \textbf{ else } S = p \times R + (1-p) \times S$$

Various laws can be proven about sequential composition. One of the most important ones, the substitution law, introduced earlier, applies to probabilistic specifications as well.

To implement a probabilistic specification we use a pseudo-random number generator. Since we cannot, even in theory, produce a real random number generator by means of traditional computing, we assume that a pseudo-random number generator generates truly random numbers and we simply refer to it as random number generator. For a positive natural variable n, we say that *rand* n produces a random natural number uniformly distributed in $0, ..n$. To reason about the values supplied by the random number generator consistently, we replace every occurrence of *rand* n with a fresh variable r whose value has probability $(r : 0, ..n)/n$. If *rand* occurs in a context such as $r = \mathit{rand}\ n$, we replace the equation by $r : (0, ..n)/n$. If *rand* occurs in the context of a loop, we parametrise the introduced variables by the execution time.

Recall the earlier example. Let us change the program slightly by introducing probabilism:

$$P \Longleftarrow \textbf{if } x = 0 \textbf{ then } \mathit{ok} \textbf{ else } (x := x - \mathit{rand}\ 2; t := t + 1; P)$$

In the new program at each iteration x is either decremented by 1 or it is unchanged, with equal probability. Our intuition tells us that the revised program should still work, except it should take longer. Let us prove it. We replace *rand* with $r : \mathit{time} \to (0, 1)$ with rt having probability $1/2$. Ignoring time we can prove:

$$\begin{aligned} & x \geq 0 \Rightarrow x' = 0 \\ \Longleftarrow\ & \textbf{if } x = 0 \textbf{ then } \mathit{ok} \textbf{ else } (x := x - \mathit{rand}\ 2; x \geq 0 \Rightarrow x' = 0) \end{aligned}$$

As for the execution time, we can prove that it takes at least x time units to complete:

$$\begin{aligned} & t' \geq t + x \\ \Longleftarrow\ & \textbf{if } x = 0 \textbf{ then } \mathit{ok} \textbf{ else } (x := x - \mathit{rand}\ 2; t := t + 1; t' \geq t + x) \end{aligned}$$

How long should we expect to wait for the execution to complete? In other words, what is the distribution of t'? Consider the following distribution of the final states:

$$(0 = x' = x = t' - t) + (0 = x' < x \leq t' - t) \times \binom{t' - t - 1}{x - 1} \times \frac{1}{2^{t'-t}},$$

$$\text{where } \binom{n}{m} = \frac{n!}{m! \times (n-m)!}$$

We can prove that:

$$\begin{aligned}
&\sum rt \cdot \frac{1}{2} \times \Big(\ \textbf{if } x = 0 \textbf{ then } ok \\
&\qquad\qquad\quad \textbf{else} \ \Big(\ x := x - rt;\ t := t + 1; \\
&\qquad\qquad\qquad\qquad (0 = x' = x = t' - t) + \\
&\qquad\qquad\qquad\qquad (0 = x' < x \leq t' - t) \ \times \ \binom{t' - t - 1}{x - 1} \times \frac{1}{2^{t'-t}} \Big) \Big) \\
&== (0 = x' = x = t' = t) \ + (0 = x' < x \leq t' - t) \times \binom{t' - t - 1}{x - 1} \times \frac{1}{2^{t'-t}}
\end{aligned}$$

Now, since for positive x, t' is distributed according to the negative binomial distribution with parameters x and $\frac{1}{2}$, its mean value is

$$\begin{aligned}
&\sum t' \cdot (t' - t) \times \left((0 = x = t' - t) + (0 < x \leq t' - t) \times \binom{t' - t - 1}{x - 1} \times \frac{1}{2^{t'-t}} \right) \\
&== \ 2 \times x + t
\end{aligned}$$

Therefore, we should expect to wait $2 \times x$ time units for the computation to complete.

3 Quantum Predicative Programming

This section is the contribution of the paper. Here we define the quantum system, introduce programming with the quantum system and several well-known problems, their classical and quantum solutions, and their formal comparative time complexity analyses. The proofs of refinements are omitted for the sake of brevity. The reader is referred to [32] for detailed proofs of some of the algorithms.

3.1 The quantum system

Let $\mathbb{C}$ be the set of all complex numbers with the absolute value operator $|\cdot|$ and the complex conjugate operator *. Then a state of an n-qubit system is a function $\psi : 0,..2^n \to \mathbb{C}$, such that $\sum x : 0,..2^n \cdot |\psi x|^2 = 1$[2].

[2] We should point out that this kind of function operations is referred to as *lifting*

If ψ and ϕ are two states of an n-qubit system, then their *inner product*, $\langle\psi|\phi\rangle : \mathbb{C}$, is defined by:

$$\langle\psi|\phi\rangle = \sum x : 0,..2^n \cdot (\psi x)^* \times (\phi x)$$

A *basis* of an n-qubit system is a collection of 2^n quantum states $b_{0,..2^n}$, such that $\forall i, j : 0,..2^n \cdot \langle b_i|b_j\rangle = (i = j)$.

We adopt the following Dirac-like notation for the computational basis: if $x : 0,..2^n$, then $\mathbf{x}$ denotes the corresponding n-bit binary encoding of x and $|\mathbf{x}\rangle : 0,..2^n \to \mathbb{C}$ is the following quantum state:

$$|\mathbf{x}\rangle = \lambda i : 0,..2^n \cdot (i = x)$$

If ψ is a state of an m-qubit system and ϕ is a state of an n-qubit system, then $\psi \otimes \phi$, the tensor product of ψ and ϕ, is the following state of a composite $m + n$-qubit system:

$$\psi \otimes \phi = \lambda i : 0,..2^{m+n} \cdot \psi(i\, div\, 2^n) \times \phi(i\, mod\, 2^n)$$

We write ${}^{\otimes n}$ to mean *tensored with itself n times.*

An operation defined on a n-qubit quantum system is a higher-order function, whose domain and range are maps from $0,..2^n$ to the complex numbers. An *identity* operation on a state of an n-qubit system is defined by

$$I^n = \lambda\psi : 0,..2^n \to \mathbb{C} \cdot \psi$$

For a linear operation A, the *adjoint* of A, written $A^\dagger$, is the (unique) operation, such that for any two states ψ and ϕ, $\langle\psi|A\phi\rangle = \langle A^\dagger\psi|\phi\rangle$. The *unitary transformations* that describe the evolution of a n-qubit quantum system are operations U defined on the system, such that $U^\dagger U = I^n$. In this setting, the *tensor product* of operators is defined in the usual way. If ψ is a state of an m-qubit system, ϕ is a state of an n-qubit system, and U and V are operations defined on m and n-qubit systems, respectively, then the tensor product of U and V is defined on an $m + n$ qubit system by $(U \otimes V)(\psi \otimes \phi) = (U\psi) \otimes (V\phi)$.

Just as with tensor products of states, we write $U^{\otimes n}$ to mean *operation U tensored with itself n times.*

Suppose we have a system of n qubits in state ψ and we measure it. Suppose also that we have a variable r from the domain $0,..2^n$, which we use to record the result of the measurement, and variables $x, y, \ldots$, which are not affected by the measurement. Then the measurement corresponds to a probabilistic specification that gives the probability distribution of ψ' and r' (these depend on ψ and on the type of measurement) and states that the variables $x, y, \ldots$ are unchanged.

For a general quantum measurement described by a collection $M = M_{0,..2^n}$ of measurement operators, which satisfy the completeness equation $\sum m : 0,..2^n \cdot M_m^\dagger M_m = I$, the specification is $\mathbf{measure}_M\, \psi\, r$, where

$$\mathbf{measure}_M\, \psi\, r == \langle\psi|M_{r'}^\dagger M_{r'}\psi\rangle \times \left(\psi' = \frac{M_{r'}\psi}{\sqrt{\langle\psi|M_{r'}^\dagger M_{r'}\psi\rangle}}\right) \times (\sigma' = \sigma)$$

where $\sigma' = \sigma$ is an abbreviation of $(x' = x) \times (y' = y) \times \ldots$ and means "all other other variables are unchanged". To obtain the distribution of, say, r' we sum out the rest of the variables as follows:

$$\sum \psi', x', y', \ldots \cdot \langle \psi | M^{\dagger}_{r'} M_{r'} \psi \rangle \times \left(\psi' = \frac{M_{r'} \psi}{\sqrt{\langle \psi | M^{\dagger}_{r'} M_{r'} \psi \rangle}} \right) \times (\sigma' = \sigma)$$

$$= \langle \psi | M^{\dagger}_{r'} M_{r'} \psi \rangle$$

For projective measurements defined by an observable $O = \sum m \cdot \lambda_m \times P_m$, where P_m is the projector on the eigenspace of O with eigenvalue λ_m:

$$\mathbf{measure}_O\, \psi\, r = \langle \psi | P_{r'} \psi \rangle \times \left(\psi' = \frac{P'_r \psi}{\sqrt{\langle \psi | P'_r \psi \rangle}} \right) \times (\sigma' = \sigma)$$

Given an arbitrary orthonormal basis $B = b_{0,..2^n}$, measurement of ψ in basis B is:

$$\mathbf{measure}_B\, \psi\, r = |\langle b_{r'} | \psi \rangle|^2 \times (\psi' = b_{r'}) \times (\sigma' = \sigma)$$

Finally, the simplest and the most commonly used measurement in the computational basis is:

$$\mathbf{measure}\ \psi\, r = |\psi r'|^2 \times (\psi' = |\mathbf{r'}\rangle) \times (\sigma' = \sigma)$$

In this case the distribution of r' is $|\psi r'|^2$ and the distribution of the quantum state is:

$$\sum r' \cdot |\psi r'|^2 \times (\psi' = |\mathbf{r'}\rangle)$$

which is precisely the mixed quantum state that results from the measurement.

In order to develop quantum programs we need to add to our list of implemented things from section 2.2. We add variables of type quantum state as above and we allow the following three kinds of operations on these variables. If ψ is a state of an n-qubit quantum system, r is a natural variable, and M is a collection of measurement operators that satisfy the completeness equation, then:

1. $\psi := |0\rangle^{\otimes n}$ is a program
2. $\psi := U\psi$, where U is a unitary transformation on an n-qubit system, is a program
3. $\mathbf{measure}_M\, \psi\, r$ is a program

The special cases of measurements, described in section 2.1, are therefore also allowed: for an observable O, $\mathbf{measure}_O\, q\, r$ is a program; for an orthonormal basis B, $\mathbf{measure}_B\, q\, r$ is a program; finally, $\mathbf{measure}\ q\, r$ is a program.

3.2 Deutsch-Jozsa algorithm

Deutsch-Jozsa problem ([14]), an extension of Deutsch's Problem ([13]), is an example of the broad class of quantum algorithms that are based on quantum Fourier transform ([23]). The task is: given a function $f : 0, ..2^n \to 0, 1$, such that f is either constant or balanced, determine which case it is. Without any restrictions on the number of calls to f, we can write the specification (let us call it S) as follows:

$$(f\ is\ constant \vee f\ is\ balanced) \Longrightarrow b' = f\ is\ constant$$

where b is a boolean variable and the informally stated properties of f are defined formally as follows:

$$\begin{aligned} &f\ is\ constant == \forall i : 0, ..2^n \cdot fi = f0 \\ &f\ is\ balanced == \left|\sum i : 0, ..2^n \cdot (-1)^{fi}\right| = 0 \end{aligned}$$

It is easy to show that

$$\begin{aligned} &\quad (f\ is\ constant \vee f\ is\ balanced) \\ &\Longrightarrow (f\ is\ constant == \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \end{aligned}$$

In our setting, we need to implement the specification R defined as follows:

$$b' == \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0$$

The *Hadamard* transform, widely used in quantum algorithms, is defined on a 1-qubit system and in our setting is a higher-order function from $0, 1 \to \mathbb{C}$ to $0, 1 \to \mathbb{C}$:

$$H = \lambda\psi : 0, 1 \to \mathbb{C} \cdot i : 0, 1 \cdot (\psi 0 + (-1)^i \times \psi 1)/\sqrt{2}$$

The operation $H^{\otimes n}$ on a n-qubit system applies H to every qubit of the system. Its action on a zero state of an n-qubit system is:

$$H^{\otimes n}|0\rangle^{\otimes n} = \sum x : 0, ..2^n \cdot |\mathbf{x}\rangle/\sqrt{2^n}$$

On a general state $|\mathbf{x}\rangle$, the action of $H^{\otimes n}$ is:

$$H^{\otimes n}|\mathbf{x}\rangle = \sum y : 0, ..2^n \cdot (-1)^{\mathbf{x}\cdot\mathbf{y}} \times |\mathbf{y}\rangle/\sqrt{2^n}$$

where $\mathbf{x} \cdot \mathbf{y}$ is the bitwise inner product of $\mathbf{x}$ and $\mathbf{y}$ modulo 2 (bitwise XOR).

Another important definition is that of the quantum analog of a classical oracle f:

$$U_f = \lambda\psi : 0, 1 \to \mathbb{C} \cdot x : 0, 1 \cdot (-1)^{fx} \times \psi x$$

The quantum solution, in one quantum variable ψ and an integer variable r is:

$$\psi := |0\rangle^{\otimes n};\ \psi := H^{\otimes n}\psi;\ \psi := U_f\psi;\ \psi := H^{\otimes n}\psi;\ \textbf{measure}\ \psi\, r;\ b := (r' = 0)$$

Let us add to the specification a restriction on the number of calls to the oracle by introducing a time variable. Suppose the new specification is:

$$(f\ \mathit{is\ constant} \vee f\ \mathit{is\ balanced} \Longrightarrow\ b' = f\ \mathit{is\ constant}) \wedge (t' = t + 1)$$

where we charge 1 unit of time for each call to the oracle and all other operations are free. Clearly, the above quantum solution works. Classically the specification is unimplementable. In fact, the strongest classically implementable specification is

$$(f\ \mathit{is\ constant} \vee f\ \mathit{is\ balanced} \Longrightarrow\ b' = f\ \mathit{is\ constant}) \wedge (t' = t + 2^{n-1} + 1)$$

3.3 Grover's search

Grover's quantum search algorithm ([18]) is well-known for the quadratic speed-up it offers in the solutions of NP-complete problems. The algorithm is optimal up to a multiplicative constant ([10]). The task is: given a function $f : 0,..2^n \to 0,1$, find $x : 0,..2^n$, such that $f x = 1$. For simplicity we assume that there is only a single solution, which we denote x_1, i.e. $f\, x_1 = 1$ and $f\, x = 0$ for all $x \neq x_1$. The proofs are not very different for a general case of more than one solutions.

As before, we use a general quantum oracle, defined by

$$U_f|\mathbf{x}\rangle = (-1)^{fx} \times |\mathbf{x}\rangle$$

In addition, we define the *inversion about mean* operator as follows:

$$M : (0,..N \to \mathbb{C}) \to (0,..N \to \mathbb{C})$$
$$M\psi == \ \lambda x : 0,..N \cdot 2 \times \left(\sum i : 0,..N \cdot \psi i/N\right) - \psi x$$

where $N = 2^n$.

Grover's algorithm initialises the quantum system to an equally weighted superposition of all basis states $|\mathbf{x}\rangle$, $x : 0,..N$. It then repeatedly applies U_f followed by M to the system. Finally, the state is measured. The probability of error is determined by the number of iterations performed by the algorithm.

The algorithm is easily understood with the help of a geometric analysis of the operators. Let α be the sum over all x, which are not solutions, and let β be the solution:

$$\alpha = \frac{1}{\sqrt{N-1}} \times \sum x \neq x_1 \cdot |\mathbf{x}\rangle$$
$$\beta = |\mathbf{x}_1\rangle$$

Then the oracle U_f performs a *reflection* about the vector α in the plane defined by α and β. In other words, $U_f(a \times \alpha + b \times \beta) = a \times \alpha - b \times \beta$. Similarly, the inversion about mean operator is a reflection about the vector ψ in the plane defined by α and β. Therefore, the result of U_f followed by M is a *rotation* in this plane. The quantum solution, in a quantum variable ψ, natural variables r, i, and k, and time variable t, is,

$$\begin{aligned}
S =\!= \ & \left(\sin\left((2 \times (t' - t) + 1) \times \arcsin \sqrt{1/N}\right)\right)^2 \times (r' = x_1) + \\
& \left(1 - \left(\sin\left((2 \times (t' - t) + 1) \times \arcsin \sqrt{1/N}\right)\right)^2\right) \times \\
& (r' \neq x_1)/(N-1) \\
=\!= \ & P;\ \textbf{measure}\ \psi\ r \\
P \Longleftarrow \ & i := 0;\ \psi := |0\rangle^{\otimes n};\ \psi := H^{\otimes n}\psi;\ R \\
R \Longleftarrow \ & \textbf{if}\ i = k\ \textbf{then}\ ok \\
& \quad \textbf{else}\ (i := i+1;\ t := t+1;\ \psi := U_f\psi;\ \psi := M\psi;\ R)
\end{aligned}$$

Specification S carries a lot of useful information. For example, it tells us that the probability of finding a solution after k iterations is

$$\left(\sin((2 \times k + 1) \times \arcsin \sqrt{1/N})\right)^2$$

Or we might ask how many iterations should be performed to minimise the probability of an error. Examining first and second derivatives, we find that the above probability is minimised when $t' - t = (\pi \times i)/(4 \times \arcsin \sqrt{1/N}) - 1/2$ for integer i. Of course, the number of iterations performed must be a natural number. It is interesting to note that probability of error is periodic in the number of iterations, but since we don't gain anything by performing extra iterations, we pick $i = 1$. Finally, assuming $1 \ll N = 2^n$, we obtain an elegant approximation to the optimal number of iterations: $\lceil \pi \times \sqrt{2^n}/4 \rceil$, with the probability of error approximately $1/2^n$.

3.4 Computing with Mixed States

As we have discussed in section 2.1, the state of a quantum system after a measurement is traditionally described as a *mixed state*. An equation $\psi = \{|0\rangle\}/2 + \{|1\rangle\}/2$ should be understood as follows: the state ψ is $|0\rangle$ with probability 1/2 and it is $|1\rangle$ with probability 1/2. In contrast to a pure state, a mixed state does not describe a physical state of the system. Rather, it describes our knowledge of in what state the system is.

In our framework, there is no need for an additional mechanism to compute with mixed states. Indeed, a mixed state is not a system state, but a distribution over system states, and all our programming notions apply to distributions. The above mixed state is the following distribution over a quantum state ψ: $(\psi = |0\rangle)/2 + (\psi = |1\rangle)/2$. This expression tells us, for each possible value in the domain of ψ, the probability of ψ having that value. For example, ψ is the state $|0\rangle$ with probability $(|0\rangle = |0\rangle)/2 + (|0\rangle = |1\rangle)/2$, which is 1/2; it is $|1\rangle$ with probability $(|1\rangle = |0\rangle)/2 + (|1\rangle = |1\rangle)/2$, which is also 1/2; for any scalars α and β, not equal to 0 or 1, ψ is $\alpha \times |0\rangle + \beta \times |1\rangle$ with probability $(\alpha \times |0\rangle + \beta \times |1\rangle = |0\rangle)/2 + (\alpha \times |0\rangle + \beta \times |1\rangle = |1\rangle)/2$, which is 0. One way to obtain this distribution is to measure an equally weighted superposition of $|0\rangle$ and $|1\rangle$:

$$\begin{aligned}
&\psi' = |0\rangle/\sqrt{2} + |1\rangle/\sqrt{2};\ \textbf{measure}\ \psi\, r \\
&&\text{measure} \\
=\!=\ &\psi' = |0\rangle/\sqrt{2} + |1\rangle/\sqrt{2};\ |\psi r'|^2 \times (\psi' = |\mathbf{r'}\rangle) \\
&&\text{sequential composition} \\
=\!=\ &\textstyle\sum r'', \psi'' \cdot (\psi'' = |0\rangle/\sqrt{2} + |1\rangle/\sqrt{2}) \times |\psi'' r'|^2 \times (\psi' = |\mathbf{r'}\rangle) \\
&&\text{one point law} \\
=\!=\ &|(|0\rangle/\sqrt{2} + |1\rangle/\sqrt{2})\, r'|^2 \times (\psi' = |\mathbf{r'}\rangle) \\
=\!=\ &(\psi' = |\mathbf{r'}\rangle)/2
\end{aligned}$$

Distribution of the quantum state is then:

$$\textstyle\sum r' \cdot (\psi' = |\mathbf{r'}\rangle)/2 =\!=\ (\psi' = |0\rangle)/2 + (\psi' = |1\rangle)/2$$

as desired.

Similarly, there is no need to extend the application of unitary operators. Consider the following toy program:

$$\psi := |0\rangle;\ \psi := H\psi;\ \textbf{measure}\ \psi\, r;\ \textbf{if}\ r = 0\ \textbf{then}\ \psi := H\psi\ \textbf{else}\ ok$$

In the second application of Hadamard the quantum state is mixed, but this is not evident from the syntax of the program. It is only in the analysis of the final quantum state that the notion of a mixed state is meaningful. The operator is applied to a (pure) system state, though we are unsure what that state is.

$$\begin{aligned}
&\psi := |0\rangle;\ \psi := H\psi;\ \textbf{measure}\ \psi\, r\,; \\
&\textbf{if}\ r = 0\ \textbf{then}\ \psi := H\psi\ \textbf{else}\ ok && \text{as before} \\
=\!=\ &(\psi' = |\mathbf{r'}\rangle)/2; \\
&\textbf{if}\ r = 0\ \textbf{then}\ \psi := H\psi\ \textbf{else}\ ok && \text{sequential composition} \\
=\!=\ &\textstyle\sum r'', \psi'' \cdot\ (\psi'' = |\mathbf{r''}\rangle)/2 \times \\
&\qquad ((r'' = 0) \times (\psi' = H\psi'') \times (r' = r'') + \\
&\qquad (r'' = 1) \times (\psi' = \psi'') \times (r' = r'')) && \text{one point law} \\
=\!=\ &((\psi' = H|0\rangle) \times (r' = 0) + (\psi' = |1\rangle) \times (r' = 1))\,/2 \\
=\!=\ &(\psi' = |0\rangle/\sqrt{2} + |1\rangle/\sqrt{2}) \times (r' = 0)/2 + \\
&(\psi' = |1\rangle) \times (r' = 1)/2
\end{aligned}$$

The distribution of the quantum state after the computation is:

$$\begin{aligned} & \sum r' \cdot (\psi' = |0\rangle/\sqrt{2} + |1\rangle/\sqrt{2}) \times (r' = 0)/2 + (\psi' = |1\rangle) \times (r' = 1)/2 \\ = {} & (\psi' = |0\rangle/\sqrt{2} + |1\rangle/\sqrt{2})/2 + (\psi' = |1\rangle)/2 \end{aligned}$$

A lot of properties of measurements and mixed states can be proven from the definitions of measurement and sequential composition. For example, the fact that a measurement in the computational basis, performed immediately following a measurement in the same basis, does not change the state of the system and yields the same result as the first measurement with probability 1, is proven as follows:

$$\begin{aligned} & \textbf{measure}\ \psi\, r;\ \textbf{measure}\ \psi\, r && \text{measure} \\ = {} & |\psi\, r'|^2 \times (\psi' = |\mathbf{r'}\rangle);\ |\psi\, r'|^2 \times (\psi' = |\mathbf{r'}\rangle) && \text{sequential composition} \\ = {} & \sum \psi'', r'' \cdot |\psi\, r''|^2 \times (\psi'' = |\mathbf{r''}\rangle) \times |\psi''\, r'|^2 \times (\psi' = |\mathbf{r'}\rangle) && \text{one point law} \\ = {} & |\psi\, r'|^2 \times (\psi' = |\mathbf{r'}\rangle) && \text{measure} \\ = {} & \textbf{measure}\ \psi\, r \end{aligned}$$

In case of a general quantum measurement, the proof is similar, but a little more computationally involved.

4 Conclusion and Future Work

We have presented a new approach to developing, analysing, and proving correctness of quantum programs. Since we adopt Hehner's theory as the basis for our work, we inherit its advantageous features, such as simplicity, generality, and elegance. Our work extends probabilistic predicative programming in the same fashion that quantum computation extends probabilistic computation. We have provided tools to write quantum as well as classical specifications, develop quantum and classical solutions for them, and analyse various properties of quantum specifications and quantum programs, such as implementability, time and space complexity, and probabilistic error analysis uniformly, all in the same framework.

Current research an research in the immediate future involve reasoning about distributed quantum computation. Current work involves expressing quantum teleportation, dense coding, and various games involving entanglement, in a way that makes complexity analysis of these quantum algorithms simple and natural. These issues will be described in a forthcoming paper. We can easily express teleportation as refinement of a specification $\phi' = \psi$, for distinct qubits ϕ and ψ, in a well-known fashion. However, we are more interested in the possibilities of simple proofs and analysis of programs involving communication, both via quantum channels and exhibiting the LOCC (local operations, classical communication) paradigm. Future work involves formalising quantum cryptographic protocols, such as BB84 [9], in our framework and provide formal analysis of these protocols. This will naturally lead to formal analysis of distributed quantum algorithms (e.g. distributed Shor's algorithm of [35]).

References

1. S. Abramsky. High-level methods for quantum computation and information. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science*, 2004.
2. S. Abramsky and B. Coecke. A categorical semantics of quantum protocols. In *LICS 2004*, 2004.
3. S. Abramsky and R. Duncan. A categorical quantum logic. In *QPL 2004*, pages 3–20, 2004.
4. P. Adao and P. Mateus. A process algebra for reasoning about quantum security. In *QPL 2005*, 2005.
5. T. Altenkirch and J. Grattage. A functional quantum programming language. In *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science*, 2005.
6. T. Altenkirch, J. Grattage, J. K. Vizzotto, and A. Sabry. An algebra of pure quantum programming. In *QPL 2005*, 2005.
7. P. Arrighi and G. Dowek. Operational semantics for formal tensorial calculus. In *QPL 2004*, pages 21–38, 2004.
8. P. Arrighi and G. Dowek. Linear-algebraic lambda-calculus. In *QPL 2005*, 2005.
9. C. H. Bennet and G. Brassard. Quantum cryptography: Public key distribution and coin tossing. In *IEEE Int. Conf. Computers, Systems and Signal Processing*, pages 175–179, 1984.
10. M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. In *Fortschritte der Physik*, pages 493–506, 1998.
11. B. Coecke. The logic of entanglement. 2004. quant-ph/0402014.
12. V. Danos, E. D'Hondt, E. Kashefi, and P. Panangaden. Distributed measurement-based quantum computation. In *QPL 2005*, 2005.
13. D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. In *Proceedings of the Royal Society of London*, pages 97–117, 1985.
14. D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London*, 439:553–558, 1992.
15. E. D'Hondt and P. Panangaden. Quantum weakest precondition. In *QPL 2004*, pages 75–90, 2004.
16. E. D'Hondt and P. Panangaden. Reasoning about quantum knowledge. 2005. quant-ph/0507176.
17. S. J. Gay and R. Nagarajan. Communicating quantum processes. In *Proceedings of the 32nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, 2005.
18. L. K. Grover. A fast quantum mechanical algorithm for database search. In *Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 212–219, 1996.
19. E.C.R. Hehner. *a Practical Theory of Programming*. Springer, New York, second edition, 2004. Available free at `www.cs.utoronto.ca/~hehner/aPToP`.
20. E.C.R. Hehner. Probabilistic predicative programming. In *Mathematics of Program Construction*, 2004.
21. E.C.R. Hehner. Retrospective and prospective for unifying theories of programming. In *Symposium on Unifying Theories of Programming*, 2006.
22. P. Jorrand and M. Lalire. Toward a quantum process algebra. In *Proceedings of the 1st ACM Conference on Computing Frontiers*, 2004.

23. R. Jozsa. Quantum algorithms and the fourier transform. *Proceedings of the Royal Society of London*, pages 323–337, 1998.
24. E. Knill. Conventions for quantum pseudocode. Technical Report LAUR-96-2724, Los Alamos National Laboratory, 1996.
25. M. Lalire and P. Jorrand. A process algebraic approach to concurrent and distributed quantum computation: operational semantics. In *QPL 2004*, pages 109–126, 2004.
26. C. Morgan and A. McIver. pQCL: formal reasoning for random algorithms. *South African Computer Journal*, 22:14–27, 1999.
27. M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.
28. B. Ömer. Quantum programming in QCL. Master's thesis, TU Vienna, 2000.
29. J. W. Sanders and P. Zuliani. Quantum programming. In *Mathematics of Program Construction*, pages 80–99, 2000.
30. P. Selinger. Towards a quantum programming language. *Mathematical Structures in Computer Science*, 2004.
31. P. Selinger. Towards a semantics for higher-order quantum computation. In *QPL 2004*, 2004.
32. A. Tafliovich. Quantum programming. Master's thesis, University of Toronto, 2004.
33. B. Valiron. Quantum typing. In *QPL 2004*, pages 163–178, 2004.
34. A. van Tonder. A lambda calculus for quantum computation. *SIAM Journal on Computing*, 33(5):1109–1135, 2004.
35. A. Yimsiriwattana and S. J. Lomonaco Jr. Distributed quantum computing: A distributed shor algorithm. 2004. `quant-ph/0403146`.
36. P. Zuliani. Non-deterministic quantum programming. In *QPL 2004*, pages 179–195, 2004.
37. P. Zuliani. Compiling quantum programs. *Acta Informatica*, 41(7-8):435–474, 2005.
38. P. Zuliani. Quantum programming with mixed states. In *QPL 2005*, 2005.